AFRL-OSR-VA-TR-2013-0520

PARALLELIZING DATA-CENTRIC PROGRAMS

JOHANNES GEHRKE

CORNELL UNIVERSITY, INC

09/25/2013
Final Report

AIR FORCE RESEARCH LABORATORY
AF OFFICE OF SCIENTIFIC RESEARCH (AFOSR)/RSL
ARLINGTON, VIRGINIA 22203
AIR FORCE MATERIEL COMMAND

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE **08 JUN 2013**	2. REPORT TYPE	3. DATES COVERED
4. TITLE AND SUBTITLE **Parallelizing Data-Centric Programs**		5a. CONTRACT NUMBER
		5b. GRANT NUMBER
		5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)		5d. PROJECT NUMBER
		5e. TASK NUMBER
		5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Cornell University,4130 Upson Hall,Ithaca ,NY,14853**		8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)		10. SPONSOR/MONITOR’S ACRONYM(S)
		11. SPONSOR/MONITOR’S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT **Approved for public release; distribution unlimited.**
13. SUPPLEMENTARY NOTES **The original document contains color images.**
14. ABSTRACT
15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**		**21**	

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std Z39-18

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)* 18-06-2013	**2. REPORT TYPE** Final Technical	**3. DATES COVERED** *(From - To)* May 2010 - May 2013

4. TITLE AND SUBTITLE
Parallelizing Data-Centric Programs

5a. CONTRACT NUMBER N/A
5b. GRANT NUMBER FA9550-10-1-0202
5c. PROGRAM ELEMENT NUMBER N/A

6. AUTHOR(S)
Gehrke, Johannes E

5d. PROJECT NUMBER N/A
5e. TASK NUMBER N/A
5f. WORK UNIT NUMBER N/A

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Cornell University, 4130 Upson Hall, Ithaca NY 14853

8. PERFORMING ORGANIZATION REPORT NUMBER
OSP 60858

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Air Force Office of Scientific Research, 875 North Randolph Street, Suite 325, Room 3112, Arlington VA 22203

10. SPONSOR/MONITOR'S ACRONYM(S)
AFOSR

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release, distribution is unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT
Increasingly, the Air Force relies on data-centric software for strategic applications. We have studied data-centric software applications on unique types of datasets -- graphs, spatial data and collections of images. As a result of our work on graphs, we have developed a new high-performance parallel graph processing framework called GRACE. For spatial data, we have conducted a comprehensive benchmarking study of existing join processing algorithms and made our benchmark available to the public to promote further improvements for this important class of algorithms. Second, we have examined the applicability of general purpose cloud infrastructure for data-centric applications. We have conducted extensive performance studies and developed a jitter-tolerant runtime for tick-based applications as well as a deployment advisor for such applications. Third, we have studied the problem of parallel agents who need to communicate and coordinate to achieve a common goal. We have developed a novel abstraction called entangled queries that allows simple and efficient coordination in a wide variety of realistic scenarios.

15. SUBJECT TERMS
data-centric software, graph processing, similarity search, entangled queries

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**	U	19	**19b. TELEPHONE NUMBER** *(include area code)*
U	U	U			607-821-1685

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std. Z39.18

Final Report: Parallelizing Data-Centric Programs

September 18, 2013

1 Overview

Increasingly, the Air Force relies on *data-centric* software for strategic applications. Data-centric software consists of programs that perform complex computations on very large datasets, in scenarios where fast and accurate performance is critical. However, maintaining this performance is becoming more and more challenging. The input datasets are continually growing larger, and the algorithms to be applied to these huge datasets are becoming more sophisticated and computationally expensive.

Until recently, we could rely on increasing clock frequency due to Moore's law to offset these growing performance demands on data-centric software. Over the last few years, however, Moore's law has hit a sharp boundary: power dissipation within a single core has reached its physical limits with current clock frequencies. Hardware continues to improve, but the main driver of performance gains is now increased *parallelism.* The number of cores on modern chips is constantly growing; also, modern cloud computing infrastructure provides programmers with multi-node clusters that are able to process parallel programs at an unprecedented scale.

In the research sponsored by this grant, we have explored a broad range of opportunities presented by parallelism for data-centric software.

First, we have studied data-centric software applications on unique types of datasets – graphs, spatial data and collections of images. As a result of our work on graphs, we have developed a new high-performance parallel graph processing framework called GRACE. For spatial data, we have conducted a comprehensive benchmarking study of existing join processing algorithms and made our benchmark available to the public to promote further improvements for this important class of algorithms. For image collections, we have studied the problem of discovering similar images within a collection and designed a number of novel algorithms to solve this problem.

Second, we have examined the applicability of general purpose cloud infrastructure for data-centric applications. We have conducted extensive performance studies and developed a jitter-tolerant runtime for tick-based applications as well as a deployment advisor for such applications.

Third, we have studied the problem of parallel agents who need to communicate and coordinate to achieve a common goal. We have developed a novel abstraction called *entangled queries* that allows simple and efficient coordination in a wide variety of realistic scenarios.

The following is a summary of the publications resulting from the work funded by this grant, arranged by project.

- **Parallel Graph Processing**: [WXDG13]
- **Benchmarking Spatial Indexing**: [SCS+14]
- **Image Processing and Parallel Similarity Search**: [ZLG11, LCG12, LSG12]
- **Data-Driven Applications in the Cloud**: [ZWS+11, CSS+11a, CSS+11b, ZBS+12]
- **Declarative Abstractions for Coordination**: [GKR+11, GKB+11, GNR+11, MOS+12]

2 Parallel Graph Processing

2.1 Combining synchronous and asynchronous models

A lot of modern data-centric software operates on datasets that represent graphs. These graphs may describe the Web, public social networks or other networks of strategic importance to organizations like the Air Force (e.g. networks related to telecommunications or other infrastructure), but they are all typically very large. Modern data-centric applications often require that complex analysis be run over these large graphs, which is computationally infeasible unless we exploit parallelism to help.

To fill the need for a parallel graph processing infrastructure, several programming frameworks have been proposed. Most of these frameworks are based on the bulk synchronous parallel (BSP) model in order to simplify application development. In the BSP model, iterative algorithms proceed by processing all graph vertices in fixed rounds until convergence. However, convergence is often slow if all vertices are processed in lockstep in each round. Asynchronous execution, in which the vertices are updated in a carefully-chosen order, often leads to much faster convergence. However, asynchronous graph processing models are typically much harder to understand and program than synchronous ones.

In our GRACE project, we have created a solution that combines the easy programmability of the BSP model with the high performance of asynchronous execution. GRACE is a new graph programming platform that separates application logic from execution policies. It provides a synchronous iterative graph programming model that enables users to easily implement, test, and debug their applications. It also contains a carefully designed and implemented parallel execution engine for both synchronous and user-specified built-in asynchronous execution policies. Our experiments show that asynchronous execution in GRACE can yield convergence rates comparable to fully asynchronous execution, while still achieving the near-linear scalability of a synchronous BSP system.

We illustrate the benefits of GRACE on a sample graph application: Image restoration (IR) for photo analysis. In this application the color of an image is captured by a large pair-wise Markov random field (MRF) with each vertex representing a pixel in the image. Belief propagation is used to compute the expectation of each pixel iteratively based on

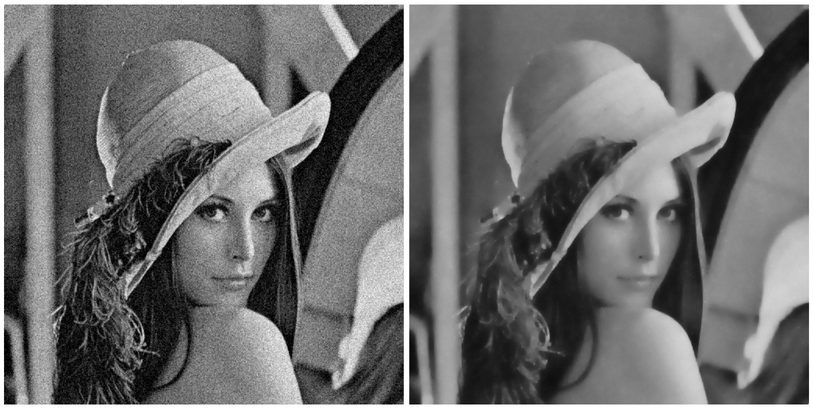


Figure 1: Qualitative Evaluation of BP Restoration on Lenna Image. Left: noisy ($\sigma = 20$). Right: restored

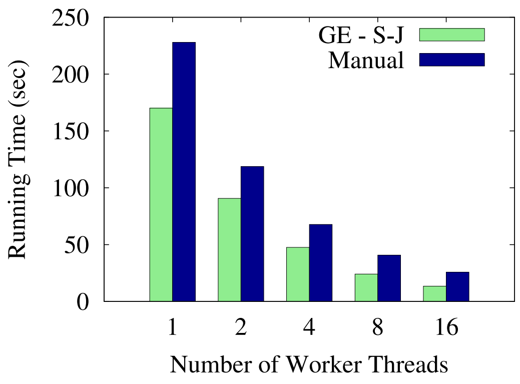


Figure 2: SfM Running Time

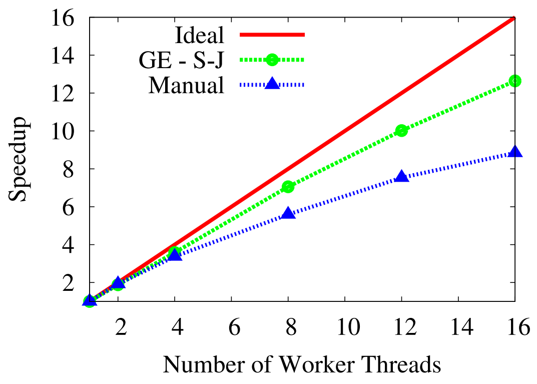


Figure 3: SfM Speedup

the observed dirty “pixel” value and weighted neighbor values. A qualitative example that shows the effectiveness of the BP algorithm is shown in Figure 1.

We have created a GRACE implementation of image restoration that is logically equivalent to a program manually written by domain experts. The performance results for up to 16 worker threads are shown in Figure 2, and the corresponding speedup results are shown in Figure 3. The algorithm reimplemented in GRACE has less elapsed time on a single CPU, illustrating that GRACE does not add significant overhead. The GRACE implementation also has better multicore speedup, in part because the manually written code estimates the absolute camera poses and the labeling error sequentially, while following the GRACE programming model this functionality is parallelized.

2.2 Increasing performance through block execution

We have also studied the important class of *computationally light* graph applications – applications that perform little computation per vertex. These applications have severe scalability problems across multiple cores, as they hit an early memory bandwidth “wall” that

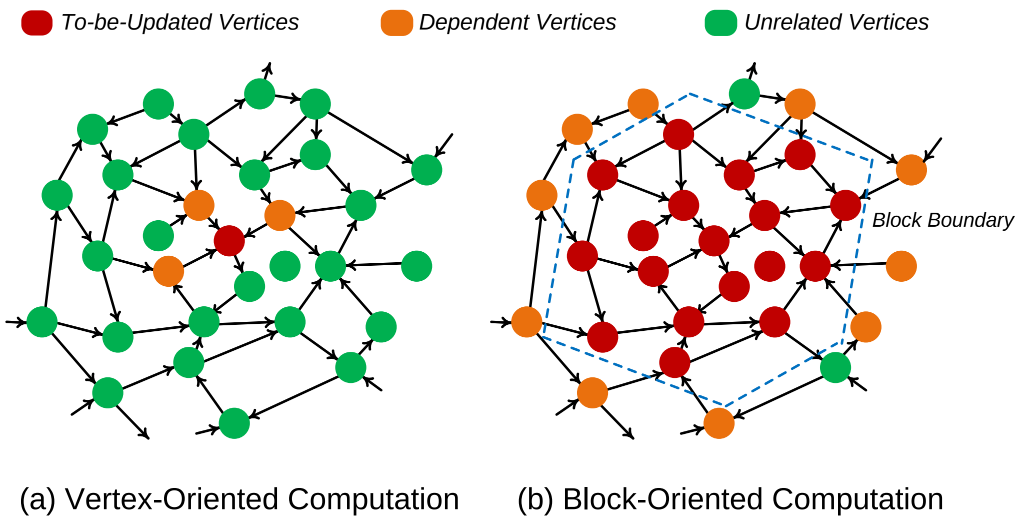


Figure 4: Vertex- vs. Block-Oriented Computation

limits their speedup. We introduced a novel block-oriented computation model in which computation is iterated locally over blocks of highly connected nodes, thus significantly improving the amount of computation per cache miss. We also designed and implemented a block-aware graph processing runtime that keeps the familiar vertex-centric programming paradigm while reaping the benefits of block-oriented execution.

Figure 4 illustrates the two computational models: Small red, yellow, and green circles represent vertices to be updated, vertices on which the update depends, and vertices unrelated to the current update. In the vertex model, only one vertex and the data on which it depends are loaded from memory for each update, while in the block processing model, all the vertices belonging to the same block are loaded from memory and updated together. For a cluster of processors, one can first partition the graph and assign subgraphs to the processors, then further partition the assigned subgraph into blocks. The subgraphs are chosen to minimize the number of edges between them, so that adjacent vertices are likely to be in the same block. Our experiments show that block-oriented execution significantly improves performance of our framework for several graph applications.

3 Benchmarking Spatial Indexing

Spatial datasets are another important use case for high-performance data-centric software. Civilian applications that use spatial data include location-based services [MK03], games [WDK$^+$07], virtual worlds [GDG$^+$09], and scientific simulations [Ver67, WSS$^+$10]. Many of these applications have counterparts in the military setting and thus are of high interest to the Air Force. In all these applications, moving objects continuously explore and sense their environment. For example, agents in games or behavioral simulations must query their surroundings in the virtual environment in order to decide what to do next.

Since many moving objects may issue similar spatial queries repeatedly, the *spatial join* is a key primitive in many of these applications. It is also common for applications such as simulations and games to batch updates in order to support evolving data more efficiently. In these applications, batches correspond naturally to the logical timesteps built into the application model. Thus many spatial applications effectively process repeated or *iterated* spatial joins, intermixed with batches of updates. We have observed that all of the example applications above, as well as many others, can be run completely in main memory. Furthermore, several recent studies have argued that many moving-object applications, including traditional location based services such as flight tracking, can tolerate some query staleness [DBVS11, ŠRJŠ11]. However, these applications require very low response times even in the presence of significant query and update rates.

In our research, we experimentally studied techniques to efficiently process iterated spatial joins for moving object applications. Determining the most efficient techniques to process these spatial joins is challenging for a variety of reasons. First, there have been a tremendous number of join algorithms proposed in the literature, many of which have never been compared directly.

Second, developers must choose a method to process updates. Since objects move through space, we can take advantage of techniques for indexing moving objects to avoid applying position updates [PCC04, vJLL00] or even update the join result itself incrementally [ISS06]. Alternatively, since updates can be batched in many moving object applications, we can either rebuild a static index or reread the data for a bulk join method.

Third, many existing algorithms for spatial joins were optimized for disk resident data since their workloads traditionally did not fit in memory. As main memory sizes continue to grow, this constraint no longer holds, so in addition to all of the decisions above, developers of high-performance spatial applications must decide which methods make sense for a main memory environment. In particular, they must consider whether memory hierarchy optimizations developed for disks translate naturally into cache optimizations for main memory.

Although previous benchmarking studies provided some guidance to developers, they fell short of comprehensively addressing the breadth of options for moving object applications. The novelty of our work lies in revisiting the vast literature on spatial joins in light of emerging workloads and commodity hardware. No previous experimental study had clarified the performance tradeoffs for the workloads we were interested in. We derived the non-obvious conclusion that if batching of queries and updates is permissible, then static methods that rebuild index structures from scratch outperform incremental methods in all but the most extreme cases. More specifically, we made the following contributions:

A Study With Many, Varied Algorithms: Our study compared ten representative techniques from the literature. We compared index nested loops joins using a variety of static and moving spatial indices as well as several special purpose spatial join algorithms. Some of these approaches had been evaluated independently in the literature, but to our knowledge no existing study had compared them all.

Experiments with Multiple Moving Object Workloads: We tuned all of the algorithms for main memory, and evaluated them on uniform and skewed random workloads as well as two workloads motivated by high-performance spatial applications: a behavioral

simulation of schooling fish [CKFL05] and a simple model of motion on road networks, which had previously been used to evaluate moving object indices [CJL08]. We experimented with a wide range of parameters so that our results can be applied to many different scenarios.
A Benchmark with Open-Source, Extensible Code: To motivate further evaluation of future and existing iterated spatial join techniques, we organized our benchmark as an extensible framework and made it available as infrastructure for the community [a]. Our APIs, code, and scripts give developers of novel algorithms easy access to an environment in which they can objectively test their algorithms against previous work. Until now, such testing has required re-implementation of previous work in a common environment.
Integration of Parallelism and Predictive Queries: In addition to comprehensively evaluating alternative methods to perform iterated spatial joins on a single processor, we also experimented with the best methods on more complex predictive queries as well as with multi-core partitioned parallelism. These additional experiments suggest directions for future research on parallel efficiency and use of static methods for even more complex query types.

4 Image Processing and Parallel Similarity Search

4.1 Image processing

Another type of dataset often processed by data-centric software is image collections, whether these come from public sites like Facebook or Flickr or from the various sensors and surveillance systems used by the Air Force.

In recent years, a number of new geometric computer vision applications have been built to reconstruct representations of landmarks using large-scale photo collections of places [FFGG$^+$10, SSS06]. A key requirement of these systems is to identify the connectivity of an image collection in the form of an *image graph* where each image is a node, and where an edge connects each pair of images that visually overlap. For unstructured image collections, the structure of this graph is initially unknown—i.e., we don't know which pairs of images match, and accordingly what edges exist—and thus needs to be discovered, usually using the tools of feature matching (e.g., with SIFT [Low04]) and RANSAC-based geometric verification [HZ03] to test the existence of edges (i.e., overlapping images). However, this verification process is relatively expensive, and so it is desirable to obtain an image graph that is as complete as possible while rigorously matching and verifying a minimal number of edges.

Given an image collection, it is often unnecessary to discover a complete description of the underlying image graph, as a much sparser graph is sufficient to capture enough information for many applications. For example, if the graph underlying a set of n images is complete (i.e., each image overlaps every other image), then discovering a star graph (one with $n-1$ edges, each connecting one node to a central node) might be a sufficient description of the graph for some applications, as compared to exhaustively matching all n choose 2 pairs of images. In the context of structure from motion (SfM), a sparse graph is often even more desirable than a complete description. On the other hand, breaking the graph into separate connected components (CCs) is undesirable. As a motivating example,

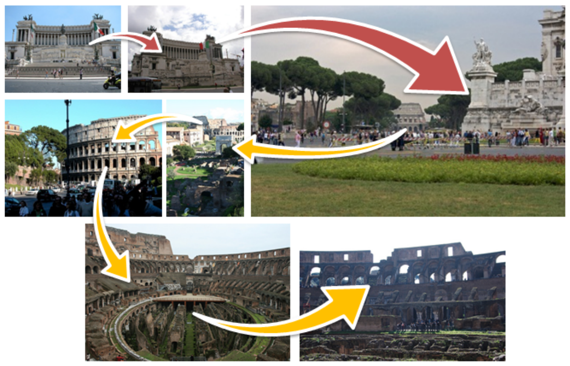


Figure 5: A visual path from Il Vittoriano to the Colosseum in our Forum dataset. Each consecutive image pair exhibits spatial overlap. In an image graph, links between two densely-connected subgraphs are important for applications such as 3D reconstruction and visualization as they contribute to a more complete scene and a smoother reconstructed model.

Figure 5 shows a "visual path" through a set of images of Rome (out of a collection of nearly 75K images) connecting two landmarks, the Il Vittoriano Monument and the Colosseum. While both these landmarks are densely photographed, photos linking the two are much more difficult to find. Hence, finding such connections is critical to linking these two monuments in a 3D model. This motivates our goal in this project: We developed ways of discovering the large CCs of an image collection as completely, and as efficiently, as possible.

We achieved our goal through a method that proposes edges to verify through feature matching and spatial verification. If each edge verification step on an image pair successfully found a match, then the problem would be much easier—we would need to test at most $n - 1$ edges to find a spanning forest for the image collection. In practice, however, it is difficult to know in advance which pairs to test, as many pairs of images do not match.

Another source of difficulty comes from the fact that large-scale image datasets often contain many (nearly) singleton images, i.e., images that match no or very few other images. Verifications on such images are wasted. Even worse, some near-singleton images contain "confusing" features that result in them being similar to other images under bag-of-words methods. Detecting such "bad" singleton images is expensive because in the worst case we have to verify a query image against all other images in the dataset to conclude that the query image is indeed a singleton. Currently, most large-scale matching systems do not explicitly model such outlier images, and hence waste computation time trying to match them.

To address the above challenges, we developed an efficient and effective algorithm called **I**mage **G**raph **M**iner (IGMiner) that applies to large-scale collections of images of scenes to discover large CCs. IGMiner works by intelligently maintaining a shortlist of image pairs to match, and re-ranking this shortlist over time based on feedback from successful and unsuccessful prior matches. We showed that a novel algorithm based on relevance feedback can be employed in IGMiner to effectively re-rank image pairs while introducing little overhead in the image retrieval process. This algorithm significantly improves the success rate of finding true matching pairs. To prevent singleton images from appearing in the shortlists of other images, we also proposed a simple yet effective measure called *rank distance* to prune out false positives and to increase the probability of success. Finally, we use an information-theoretic model of the problem to choose edges that minimize expected entropy given estimated prior probabilities of matches based on visual similarities.

We demonstrated the effectiveness of IGMiner on several image collections with ground truth obtained by exhaustive geometric verification, as well as larger image collections with tens or hundreds of thousands of images. Our experiments showed that IGMiner can effectively identify large CCs in an image graph with a relatively small number of matching operations, and that for large problems it can produce significantly better results than current techniques, such as ImageWebs [HGO$^+$10], given the same budget of matches performed.

4.2 Scalable Parallel Similarity Search

The work described above spurred us to develop solutions for another problem that has applications far beyond image processing, namely, similarity search. Finding similar objects is a very common problem, occurring in domains such as document and image clustering [CM10, HGI00], plagiarism detection [SGWG06], near duplicate documents detection [Hen06], 3D scene reconstruction [ASS$^+$09], similar music and video retrieval [FML04, TBTL07], community mining [SSB05], and personalized recommendations [DDGR07]. Again, many of these applications in the civilian world have military counterparts and the ability to perform fast similarity search is critical to the needs of the Air Force.

A natural way to describe complex objects is to represent them as vectors in a high dimensional space, where the dimensions correspond to the features extracted from the objects. For example, there are hundreds of thousands of words in an unabridged English dictionary, and usually each word is considered a feature. Standard content-based image retrieval algorithms preprocess images by extracting local features [Low04] and quantizing them into visual words [NS06]; research shows that retrieval systems using a million visual words tend to outperform those using a smaller visual vocabulary [NS06]. Large-scale recommendation systems need to find the similarities of users over millions of items [DDGR07]. In many important applications, these vectors are either binary, or can be approximated by binary vectors.

In this project, we introduced a novel concept for similarity search that is based on the use of *random filters*. Algorithms built on this idea repeatedly apply random filters to discard irrelevant vector pairs; thus, we call them *filtering-based* algorithms. We showed that we can use truly random permutations to cheaply generate each random filter, then use inverted indexes to compute the intersection between the outputs of all filters to arrive at the candidate set. In addition, we demonstrated that truly random permutations can also be used to reliably estimate the similarity between vector pairs. This leads to a new candidate filtering algorithm, which works especially well for objects whose vector representation typically contains many non-zero elements, such as text, images, and videos.

Finally, we observed that cluster-like structures abound in real data; text documents can be described with a set of topics, and photographs are often taken near objects of interests. We proposed a clustering algorithm that exploits this property for further speed-up.

The main contributions of this project were as follows:

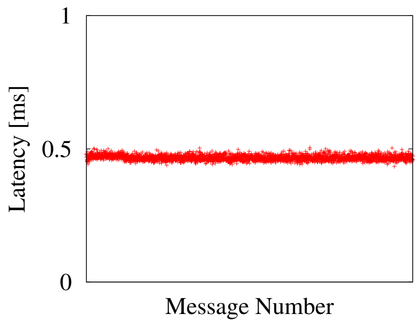
- We proposed a filtering-based algorithm for generating vector pairs whose similarities are likely to be above the given threshold. This algorithm significantly outperforms other candidate generation algorithms for low similarity thresholds, supports incremental queries, and can easily be parallelized.
- We presented an efficient algorithm for estimating the similarities between vector pairs. We showed analytically that we only need to examine a fixed fraction of the dimensions in order to discard irrelevant vector pairs.
- We presented a fast approximate clustering algorithm that exploits cluster structures in real data. Our algorithm uses random sampling and probabilistic assignments, and can reduce the search space by up to 92% while discarding less than 1% of the true positives.
- Based on these three ideas, we developed a novel probabilistic similarity search algorithm called ATLAS, and we provide experimental results on several real-world datasets. At a 97.5% recall rate, ATLAS is up to 210 times faster than previous exact algorithms and up to 80 times faster than previous approximate algorithms.

5 Data-Driven Applications in the Cloud

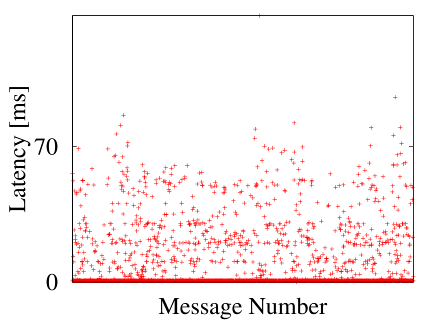
In this project, we investigated what happens when data-centric software is moved from expensive custom HPC centers and private clusters into the cheaper but much less controlled environment of the public cloud. Our results are important for any organization considering the pros and cons of using more economical cloud infrastructure for their data-centric needs. The Air Force, of course, has aspects beyond raw performance to consider, such as data security; however, performance is also a factor whose importance cannot be discounted.

Our specific focus in this project was on scientific simulations that make frequent use of synchronization barriers. Because of these barriers, the code is very sensitive to fluctuations in performance. As a consequence, most modern HPC centers allocate whole portions of a cluster exclusively for execution of an application. This model works well for heavy science users, but is not ideal for mid-range applications that only need to use a few hundred compute nodes. In particular, these mid-range users have to wait on execution queues for long periods–sometimes hours or even days–to get to run their jobs. This significantly lengthens the time-to-solution.

Our work examined what happens when we take these scientific applications off those private, well-behaved, expensive computing platforms and run them in the cloud.

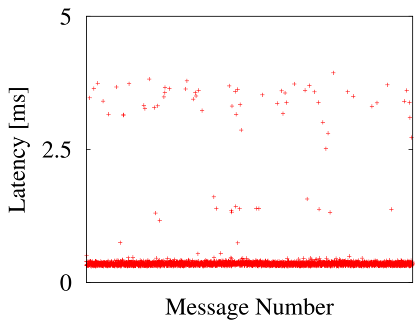


(a) Weblab Instances

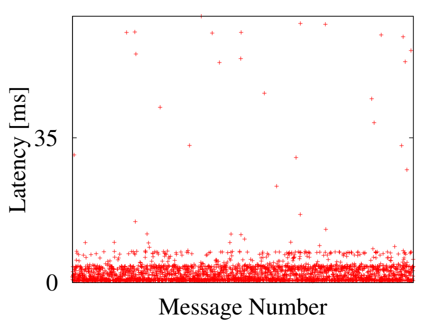


(b) EC2 Small Instances

Figure 6: Latency in Our Weblab Cluster and in EC2 Small Instances



(a) EC2 Cluster Instances



(b) EC2 Large Instances

Figure 7: Latency in EC2

5.1 Making Time-Stepped Applications Tick in the Cloud

Many data-centric cloud applications proceed in discrete time-steps or "ticks," using communication barriers at tick boundaries to synchronize computation. One critical assumption which does not hold in the cloud is that there is a stable, low-latency interconnect among compute nodes. Recent experimental studies have demonstrated that the cloud suffers from high latency jitter. We confirmed this observation by measuring the TCP round-trip times for 16 KB messages in several environments, as shown in Figure 7. These environments include the Cornell Weblab, which is a modest dedicated cluster of machines interconnected by Gigabit Ethernet, and Amazon EC2 cloud instances in the 32-bit "Small", 64-bit "Large" and 64-bit "Cluster Compute" categories. Note that the scales of the y-axes differ significantly. Communication in the Weblab is well-behaved, with latencies tightly distributed around the mean. The 32-bit EC2 instances have poor performance, with high average latency and high variance. The 64-bit EC2 instance categories show acceptable average latency, but suffer frequent latency "spikes" more than an order of magnitude above the mean. Even the cluster compute instances, advertised for HPC applications, show the same effect. Unfortunately, network jitter can severely increase the time required for synchronization barriers, significantly increasing overall running time.

To alleviate the above problem, we developed a general, jitter-tolerant runtime to process time-stepped scientific applications in the cloud. Our runtime exposes a high-level, data-centric API to the scientist, who designates application state as tables and dependencies between state as queries over these tables. Our runtime uses these data dependencies to (1) carefully schedule computation and (2) replicate data and computation to absorb latency spikes. Our data-driven approach is completely transparent to the scientist and requires little additional code. Our experimental results show that our methods improve performance up to a factor of three for several typical scientific applications.

5.2 ClouDiA: A Deployment Advisor for Public Clouds

Iin the ClouDiA project, we expanded on the above work to create an automated advisor for deploying time-step based applications in the public cloud. ClouDiA selects application node deployments minimizing either (i) the largest latency between application nodes, or (ii) the longest critical path among all application nodes. ClouDiA employs mixed-integer programming and constraint programming techniques to efficiently search the space of possible mappings of application nodes to instances.

How applications are deployed on a public cloud can strongly affect performance. To achieve high utilization, cloud providers inevitably allocate virtual machine instances non-contiguously, i.e., instances of a given application may end up in physically distant machines in the cloud. This allocation strategy can lead to large differences in average latency between instances. For a large class of applications, this difference can result in significant performance degradation, unless care is taken in how application components are mapped to instances.

Latency-sensitive applications in the cloud can be classified into two broad classes: high-performance computing applications, for which the main performance goal is *time-to-solution*, and service-oriented applications, for which the main performance goal is *response time for service calls.*

Goal: Time-to-solution. A number of HPC applications simulate natural processes via long-running, distributed computations. At every time step of the simulation, neighboring nodes exchange messages before proceeding to the next time step. As the end of a time step is a logical barrier, worst-link latency essentially determines communication cost [AP97, BJvOR03, KHJ98, ZWS^{+}11] and time-to-solution is dramatically affected by the latency of the worst link.

Goal: Service response time. Web services and portals, as well as search engines, are prime cloud applications [AWS, GGL09, Vog07]. The rendering of a web page in these portals is the result of tens or hundreds, of web service calls [O'H06]. While different portions of the web page can be constructed independently, there is still a critical path of service calls that determines the server-side communication time to respond to a client request. Latencies in the critical path add up, and can negatively affect end-user response time.

Figure 8 depicts the architecture of ClouDiA. The dashed line indicates the boundary between ClouDiA and public cloud tenants. The tuning methodology followed by ClouDiA comprises the following steps:

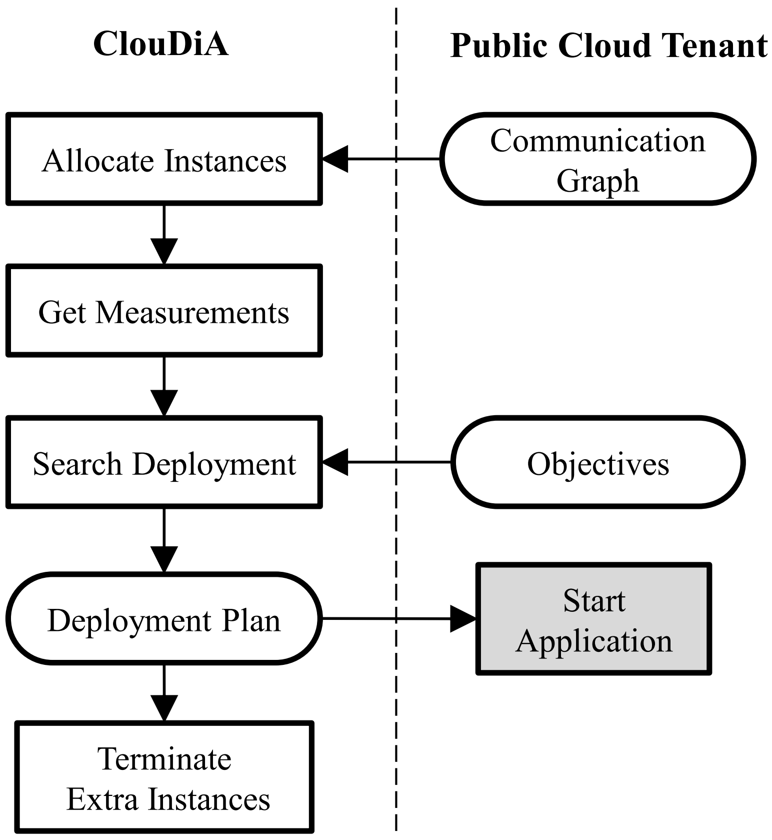


Figure 8: Architecture of ClouDiA

1. Allocate Instances: A tenant specifies the communication graph for the application, along with a maximum number of instances at least as great as the required number of application nodes. CloudDiA then automatically allocates cloud instances to run the application. Depending on the specified maximum number of instances, ClouDiA will over-allocate instances to increase the chances of finding a good deployment.

2. Get Measurements: The pairwise latencies between instances can only be observed after instances are allocated. ClouDiA performs efficient network measurements to obtain these latencies. The main challenge is reliably estimating the mean latencies quickly, given that time spent in measurement is not available to the application.

3. Search Deployment: Using the measurement results, together with the optimization objective specified by the tenant, CloudDiA searches for a "good" deployment plan: one that avoids "bad" communication links. We formalize this notion and pose the node deployment problem, and we then formulate two variants of the problem that model our two classes of latency-sensitive applications. Given the hardness of these problems, traditional methods cannot scale to realistic sizes. We propose techniques that significantly speed up this search.

4. Terminate Extra Instances: Finally, ClouDiA terminates any over-allocated instances and the tenant can start the application with an optimized node deployment plan.

Through experiments with synthetic and real applications in Amazon EC2, we showed that our techniques yield a 15% to 55% reduction in time-to-solution or service response time, without any need for modifying application code.

6 Declarative Abstractions for Coordination

The final project we carried out under this grant related to language abstractions for synchronization and coordination. While the other projects demonstrated how to achieve substantial performance gains through parallelism, this project explored how parallel agents can work together, communicate and coordinate to achieve common goals. Coordination is a concept with many applications, whether for social activities such as choosing a time to see a movie with friends or in more serious applications in line with the typical use cases of the Air Force, where multiple agents in the field need to coordinate to achieve a military goal.

While the need for coordination is pervasive, coordination is not commonly supported by today's data-related abstractions and is typically achieved through ad-hoc methods (such as phone calls to organize joint travel plans, followed by a period where everyone tries to make flight bookings simultaneously and hopes enough seats are available).

In this project, we introduced and developed abstractions for performing coordination in a principled and mathematically organized manner. We focused on coordination related to data of some sort, for example the location of a joint trip or a joint tactical strike; we call this kind of coordination *data-driven coordination* and we developed declarative languages and mechanisms to facilitate it. The main idea is to provide a way for users to coordinate within the system without having to worry about the details of the coordination. Because the coordination is data-driven, the coordination abstraction is designed to sit at the same level as other abstractions that relate to the data. Declarativity – allowing users to express what is to be achieved, rather than how it is to be achieved – has long been an underlying design principle in databases. In a declarative specification of coordination, the users' only responsibility is to state their individual preferences and constraints, and the system takes care of the rest.

Our main declarative primitive for data-driven coordination is *entangled queries*; these are special kinds of queries that database users can pose which express their desire to coordinate on other values. We have developed the entangled query model in detail, shown how these queries can be embedded in larger code units such as transactions, and studied the complexity of the evaluation of these queries.

6.1 Entangled Queries

To see what coordination looks like in a system that supports entangled queries, consider an example. Suppose Kramer wants to travel to Paris on the same flight as Jerry. In our system, he can express his request with the following *entangled query*:

```
SELECT ‘Kramer’, fno INTO ANSWER Reservation
WHERE
fno IN (SELECT fno FROM Flights WHERE dest=‘Paris’)
```

```
AND ('Jerry', fno) IN ANSWER Reservation
CHOOSE 1
```

Jerry also wants to travel with Kramer, but he has an additional constraint: he wants to travel only on flights operated by United. His query is as follows:

```
SELECT 'Jerry', fno INTO ANSWER Reservation
WHERE
fno IN (SELECT fno FROM Flights F, Airlines A WHERE
                   F.dest='Paris' and F.fno = A.fno
                   AND A.airline = 'United' )
AND ('Kramer', fno) IN ANSWER Reservation
CHOOSE 1
```

For this report, it is enough to understand that `Reservation` is a name for a virtual relation that contains the answers to all the current queries in the system. The `SELECT` clause specifies Kramer's own expected answer, or, in other words, his contribution to the answer relation `Reservation`. This contribution, however, is conditional on two requirements, which are given in the `WHERE` clause. First, the flight number in question must correspond to a flight to Paris. Second, the answer relation must also contain a tuple with the same flight number but `Jerry` as the traveler name. Jerry's query places a near-symmetric constraint on `Reservation`.

Neither user explicitly specifies which other queries he wishes to coordinate with – e.g. by using an identifier for the coordination partner's query. Instead, the coordination partner is designated implicitly using the partner's query result. This is a deliberate choice that allows coordination with potentially unknown partners based purely on desired shared outcomes. In travel planning, of course, it typically *is* known who one's coordination partners will be. However in other scenarios such as MMO games, coordination partners may be unknown and their identities irrelevant.

When the system receives Kramer and Jerry's queries, it answers both of them simultaneously in a way that ensures a coordinated flight number choice. In general, there may be many different suitable flights, but Kramer and Jerry only want to make a booking on one of them. The `CHOOSE 1` clause present in both queries specifies that only one tuple is to be returned per query. The tuples returned must be such that all constraints are satisfied. If the database is as shown in Figure 9 (a), the system non-deterministically chooses either flight 122 or 123 and returns appropriate answer tuples. Figure 9 (b) shows the mutual constraint satisfaction that takes place in answering for 122. The intent is that Kramer and Jerry should now be able to make a booking on the particular flight mentioned in the query answer.

The above queries are of course simplified to illustrate the basic coordination mechanic; in a real travel reservation setting, they would include checks for seat availability and other factors.

In our work, we formalized entangled queries, extensively studied the complexity of their evaluation, designed efficient evaluation algorithms for cases where efficient evaluation is possible, and implemented an end-to-end system that supports them. Experimental results

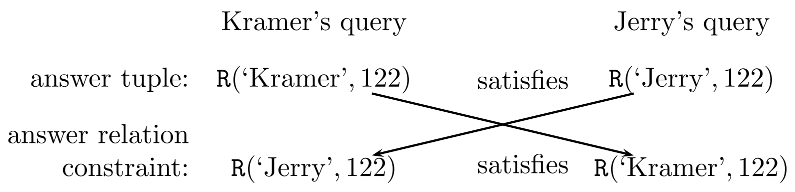
Flights

fno	dest
122	Paris
123	Paris
134	Paris
136	Rome

Airlines

fno	airlines
122	United
123	United
134	Lufthansa
136	Alitalia

(a)



(b)

Figure 9: (a) Flight database (b) Mutual constraint satisfaction

confirm that our evaluation algorithms are practical on workloads based on real social network data.

6.2 Entangled Transactions

Entangled queries are a useful building block for coordination. However, most real-world data management applications that involve coordination require not just queries, but a transaction-like abstraction that covers larger units of work. As an example, assume that two friends, Mickey and Minnie, wish to travel to Los Angeles on the same flight and stay at the same hotel. Their arrival date is flexible, but their departure date is fixed. They start by jointly selecting a suitable flight. Once they know the flight number, and consequently their date of arrival in Los Angeles, they will try to make appropriate joint hotel reservations. As explained previously, they can use entangled queries to coordinate on the choice of the flight and then on their choice of hotel. These queries, however, need to be embedded within a larger code unit that Mickey and Minnie separately execute and populate with their constraints such as the class of the hotel or airline restrictions. Once both of their individual *entangled transactions* have been submitted, the system needs to match up these two transactions, execute the associated logic, and guarantee "transaction-like" semantics for this execution.

In our work, we introduced a model of entangled transactions that comes with analogues of the classical ACID properties. We considered multiple possible execution models for entangled transactions and implemented a prototype execution engine. Experiments with our prototype show that the overheads associated with supporting entangled transactions are acceptable for real-world use.

References

[a] Cornell Database Group Website. `www.cs.cornell.edu/bigreddata/`.

[AP97] Richard D. Alpert and James F. Philbin. cBSP: Zero-cost synchronization in a modified BSP model. Technical report, NEC Research Institute, 1997.

[ASS+09] Sameer Agarwal, Noah Snavely, Ian Simon, Steven M. Seitz, and Richard Szeliski. Building Rome in a day. In *ICCV*, 2009.

[AWS] Amazon web services, search engines & web crawlers. http://aws.amazon.com/search-engines/.

[BJvOR03] Olaf Bonorden, Ben Juurlink, Ingo von Otte, and Ingo Rieping. The paderborn university BSP (PUB) library. *Parallel Computing*, 29(2):187–207, 2003.

[CJL08] Su Chen, Christian S. Jensen, and Dan Lin. A benchmark for evaluating moving object indexes. *Proc. VLDB*, pages 1574–1585, 2008.

[CKFL05] I. D. Couzin, J. Krause, N. R. Franks, and S. A. Levin. Effective leadership and decision-making in animal groups on the move. *Nature*, 433(7025):513–516, 2005.

[CM10] Ondrej Chum and Jiri Matas. Large-scale discovery of spatially related images. *IEEE Trans. Pattern Anal. Mach. Intell.*, 32(2):371–377, 2010.

[CSS+11a] Tuan Cao, Marcos Antonio Vaz Salles, Benjamin Sowell, Yao Yue, Alan J. Demers, Johannes Gehrke, and Walker M. White. Fast checkpoint recovery algorithms for frequently consistent applications. In *SIGMOD Conference*, pages 265–276, 2011.

[CSS+11b] Tuan Cao, Benjamin Sowell, Marcos Antonio Vaz Salles, Alan J. Demers, and Johannes Gehrke. Brrl: a recovery library for main-memory applications in the cloud. In *SIGMOD Conference*, pages 1233–1236, 2011.

[DBVS11] Jens Dittrich, Lukas Blunschi, and Marcos Antonio Vaz Salles. MOVIES: indexing moving objects by shooting index images. *Geoinformatica*, 15(4):727–767, 2011.

[DDGR07] Abhinandan S. Das, Mayur Datar, Ashutosh Garg, and Shyam Rajaram. Google news personalization: scalable online collaborative filtering. In *WWW*, 2007.

[FFGG+10] J.M. Frahm, P. Fite-Georgel, D. Gallup, T. Johnson, R. Raguram, C. Wu, Y.H. Jen, E. Dunn, B. Clipp, S. Lazebnik, et al. Building rome on a cloudless day. In *ECCV*, 2010.

[FML04] S. L. Feng, R Manmatha, and V. Lavrenko. Multiple bernoulli relevance models for image and video annotation. In *CVPR*, 2004.

[GDG$^+$09] N. Gupta, A. Demers, J. Gehrke, P. Unterbrunner, and W. White. Scalability for virtual worlds. In *Proc. ICDE*, 2009.

[GGL09] Roxana Geambasu, Steven D. Gribble, and Henry M. Levy. Cloudviews: Communal data sharing in public clouds. In *HotCloud*, 2009.

[GKB$^+$11] Nitin Gupta, Lucja Kot, Gabriel Bender, Sudip Roy, Johannes Gehrke, and Christoph Koch. Coordination through querying in the youtopia system. In *SIGMOD Conference*, pages 1331–1334, 2011.

[GKR$^+$11] Nitin Gupta, Lucja Kot, Sudip Roy, Gabriel Bender, Johannes Gehrke, and Christoph Koch. Entangled queries: enabling declarative data-driven coordination. In *SIGMOD Conference*, pages 673–684, 2011.

[GNR$^+$11] Nitin Gupta, Milos Nikolic, Sudip Roy, Gabriel Bender, Lucja Kot, Johannes Gehrke, and Christoph Koch. Entangled transactions. *PVLDB*, 4(11):887–898, 2011.

[Hen06] Monika Rauch Henzinger. Finding near-duplicate web pages: a large-scale evaluation of algorithms. In *SIGIR*, 2006.

[HGI00] Taher H. Haveliwala, Aristides Gionis, and Piotr Indyk. Scalable techniques for clustering the web. In *WebDB*, 2000.

[HGO$^+$10] Kyle Heath, Natasha Gelfand, Maks Ovsjanikov, Mridul Aanjaneya, and Leonidas J. Guibas. Image webs: Computing and exploiting connectivity in image collections. In *CVPR*, 2010.

[HZ03] Richard Hartley and Andrew Zisserman. *Multiple View Geometry in Computer Vision.* Cambridge University Press, 2nd edition, 2003.

[ISS06] Glenn S. Iwerks, Hanan Samet, and Kenneth P. Smith. Maintenance of k-nn and spatial join queries on continuously moving points. *ACM Trans. Database Syst.*, 31(2):485–536, 2006.

[KHJ98] Jin-Soo Kim, Soonhoi Ha, and Chu Shik Jhon. Efficient barrier synchronization mechanism for the BSP model on message-passing architectures. In *IPPS/SPDP*, 1998.

[LCG12] Yin Lou, Rich Caruana, and Johannes Gehrke. Intelligible models for classification and regression. In *KDD*, pages 150–158, 2012.

[Low04] David G. Lowe. Distinctive image features from scale-invariant keypoints. *IJCV*, 60(2), 2004.

[LSG12] Yin Lou, Noah Snavely, and Johannes Gehrke. Matchminer: Efficient spanning structure mining in large image collections. In *ECCV (2)*, pages 45–58, 2012.

[MK03] Jussi Myllymaki and James Kaufman. High-performance spatial indexing for location-based services. In *Proc. WWW*, pages 112–117, 2003.

[MOS+12] Konstantinos Mamouras, Sigal Oren, Lior Seeman, Lucja Kot, and Johannes Gehrke. The complexity of social coordination. *PVLDB*, 5(11):1172–1183, 2012.

[NS06] David Nistér and Henrik Stewénius. Scalable recognition with a vocabulary tree. In *CVPR*, 2006.

[O'H06] Charlene O'Hanlon. A conversation with werner vogels. *ACM Queue*, 4(4):14–22, May 2006.

[PCC04] Jignesh M. Patel, Yun Chen, and V. Prasad Chakka. STRIPES: an efficient index for predicted trajectories. In *Proc. SIGMOD*, pages 635–646, 2004.

[SCS+14] Ben Sowell, Tuan Cao, Marcos Vaz Salles, Alan Demers, and Johannes Gehrke. An experimental analysis of iterated spatial joins in main memory. In *Proceedings of the VLDB Endowment (to appear, accepted for publication)*, 2014.

[SGWG06] Daria Sorokina, Johannes Gehrke, Simeon Warner, and Paul Ginsparg. Plagiarism detection in arXiv. In *ICDM*, 2006.

[ŠRJŠ11] Darius Šidlauskas, Kenneth Ross, Christian Jensen, and Simonas Šaltenis. Thread-level parallel indexing of update intensive moving-object workloads. In *Advances in Spatial and Temporal Databases*, volume LNCS 6849, pages 186–204. Springer Berlin / Heidelberg, 2011.

[SSB05] Ellen Spertus, Mehran Sahami, and Orkut Buyukkokten. Evaluating similarity measures: a large-scale study in the Orkut social network. In *KDD*, 2005.

[SSS06] Noah Snavely, Steven M. Seitz, and Richard Szeliski. Photo tourism: exploring photo collections in 3d. *ACM Trans. Graph.*, 25(3):835–846, 2006.

[TBTL07] Douglas Turnbull, Luke Barrington, David Torres, and Gert Lanckriet. Towards musical query-by-semantic-description using the CAL500 data set. In *SIGIR*, 2007.

[Ver67] Loup Verlet. Computer "experiments" on classical fluids. i. thermodynamical properties of lennard-jones molecules. *Phys. Rev.*, 159(1):98, 1967.

[vJLL00] Simonas Šaltenis, Christian S. Jensen, Scott T. Leutenegger, and Mario A. Lopez. Indexing the positions of continuously moving objects. In *Proc. SIGMOD*, pages 331–342, 2000.

[Vog07] Werner Vogels. Data access patterns in the amazon.com technology platform. In *VLDB*, page 1, 2007.

[WDK+07] W. White, A. Demers, C. Koch, J. Gehrke, and R. Rajagopalan. Scaling games to epic proportions. In *Proc. SIGMOD*, 2007.

[WSS+10] Guozhang Wang, Marcos Vaz Salles, Benjamin Sowell, Xun Wang, Tuan Cao, Alan Demers, Johannes Gehrke, and Walker White. Behavioral simulations in mapreduce. *Proc. VLDB*, 3:952–963, September 2010.

[WXDG13] Guozhang Wang, Wenlei Xie, Alan J. Demers, and Johannes Gehrke. Asynchronous large-scale graph processing made easy. In *CIDR*, 2013.

[ZBS+12] Tao Zou, Ronan Le Bras, Marcos Antonio Vaz Salles, Alan J. Demers, and Johannes Gehrke. Cloudia: A deployment advisor for public clouds. *PVLDB*, 6(2):109–120, 2012.

[ZLG11] Jiaqi Zhai, Yin Lou, and Johannes Gehrke. Atlas: a probabilistic algorithm for high dimensional similarity search. In *SIGMOD Conference*, pages 997–1008, 2011.

[ZWS+11] Tao Zou, Guozhang Wang, Marcos Vaz Salles, David Bindel, Alan Demers, Johannes Gehrke, and Walker White. Making time-stepped applications tick in the cloud. In *SOCC*, 2011.